AFRL-IF-RS-TR-1999-29
Final Technical Report
February 1999

HYPERMEDIA PROTOTYPE FOR DEMONSTRATION

Georgia Tech Research Institute

Susan Liebeskind, Jay D. Bolter, and Kirk Pennywitt

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

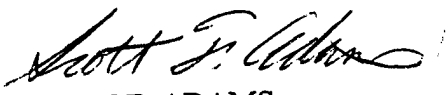
19990323 055

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

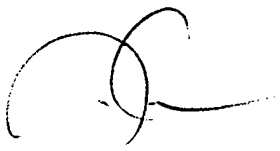
DTIC QUALITY INSPECTED 1

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-29 has been reviewed and is approved for publication.

APPROVED: 

SCOTT F. ADAMS
Project Engineer

FOR THE DIRECTOR: 

JOSEPH CAMERA, Deputy Chief
Information & Intelligence Exploitation Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFEC, 32 Brooks Road, Rome, NY 13441-4114. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1999	Final Sep 95 - Dec 97

4. TITLE AND SUBTITLE

HYPERMEDIA PROTOTYPE FOR DEMONSTRATION

5. FUNDING NUMBERS

C - F30602-95-C-0124
PE - 63726F
PR - 2810
TA - BA
WU - 3B

6. AUTHOR(S)

Susan Liebeskind, Jay D. Bolter, and Kirk Pennywitt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Georgia Tech Research Institute
Georgia Institute of Technology, Computer Science & Information Technology Laboratory
Atlanta GA 30332

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/IFEC
32 Brooks Road
Rome NY 13440-4114

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-29

11. SUPPLEMENTARY NOTES

Air Force Research Laboratory Project Engineer: Scott F. Adams/IFEC/(315) 330-1430

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report is the culmination of development of HyperTech, an interface to multimedia data (text, graphics, imagery, video, and audio). Hypermedia capitalized upon the associations between data hypermedia can be used to produce large, richly-connected and cross-referenced bodies of information. The software has multiple methods of viewing data, automated lining based upon text content, and import/export of Hypertext Markup Language (HTML) format. The environment was built upon a relational database. The payoff is an alternative interface which can supplement existing retrieval methods. This work focused upon improving access to data on intelligence workstations. This report provides an overview of the effort, a listing of applicable documents, detail discussions of the approach, and suggestions for future work.

14. SUBJECT TERMS

Hypermedia, Hypertext, Database, Multimedia

15. NUMBER OF PAGES

44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Hypermedia Prototype for Demonstration
(HyperTech v3.0)
Final Technical Report

TABLE OF CONTENTS

1. Introduction 1
1.1 HyperTech History 1
1.2 Compatibility with HyperTech 2.0 2
1.3 HyperTech 3.0 Functionality 2
1.4 Reference Documents 11

2. Detailed Discussion 12
2.1 Multi-user Support 12
2.2 Use of Xew Widget Set 12
2.3 HyperTech Storage Format 13
2.4 Enhancement of Navigation Metaphor 14
2.5 Data Acquisition 14
2.6 Automatic Path Generation 14
2.7 User Interface Techniques and Usability Testing 14
2.8 Design of User Interface and Data Model 15
2.9 Object-Oriented Software Development 15
2.10 Performance Issues 16
2.11 Port to Solaris 16
2.12 HTML Export and Import 16

3. Areas for Potential Future Development 19
3.1 Enhanced Multi-user Support Features 19
3.2 User Interface Toolkit Alternatives 19
3.3 Data Storage Alternatives 19
3.4 User Interface Enhancements 20
3.5 Porting to the Java Language 20
3.6 Performance Enhancements 21

4. Conclusions 22

APPENDIX A – Selections from HyperTech v2.0 Final Technical Report 24

APPENDIX B – Usability Testing Recommendations 35

APPENDIX C – Acronym List 36

1. Introduction

Modern intelligence analysis and information management tasks require the capability to administer and manipulate large volumes of text, imagery, graphics and video data. A hypertext system is often the most effective solution for storing and presenting complex and interrelated information. To provide such a solution, the HyperTech application was developed by the Georgia Institute of Technology and the Georgia Tech Research Institute (GTRI) under contract to the US Air Force Rome Laboratory/IRRE.

Key hypermedia concepts include: division of information into well-defined units; connection of these units with intuitive, navigable links; interactive, flexible, non-linear access to the resulting information network, or "web". Potential benefits include increased ease in: finding relevant information, adding/integrating information, presenting information, customizing information, and selecting subsets of data.

This Final Technical Report discusses lessons learned in designing, implementing and porting HyperTech. It provides an analysis of the elements of HyperTech which might be improved through redesign or reimplementation, or which were impacted by circumstances beyond the developers' control. Lastly we provide some recommendations for enhancements to, and future directions for, the HyperTech program.

1.1 HyperTech History

This work was initially proposed by an Air Combat Command background paper "Integration of Hypermedia with Intelligence Data Bases", February 1992. This paper related the deficiency that automated tools provided only a portion of the capabilities needed by unit-level intelligence personnel. Research and reference materials were still largely available only in hardcopy form. Hardcopy problems create mobility problems since safes are required and are not easily deployable. Reference materials need to be digitized and installed on unit-level intelligence systems. Digitized reference material needs to be tied to appropriate data base elements, not just incorporated as an electronic "page turner" (i.e. if data base queries reveal a certain aircraft is at a particular airfield, a window should appear with appropriate aircraft performance information). Since most commonly used reference materials are revised on an annual or semi-annual basis, such hypermedia products must be easy to update.

Development of HyperTech began before the explosive growth of the Internet, triggered by the release of Mosaic, the first browser, in 1993. The subsequent rapid evolution of the Internet and browser technology provides evidence of the utility of hypertext/hypermedia for communication and information access. Indeed, if development of HyperTech had commenced only a short time later, browsers would likely have played a much larger role.

The HyperTech project began in October 1993. HyperTech 1.0, for SunOS 4.1.3, was delivered to Rome Laboratory in March 1995. It included the major HyperTech views and helper dialogs, support for a single user of the database, and the ability to export HyperTech data to HTML format. HyperTech 2.0, also for SunOS 4.1.3, was delivered in May 1996. It added multi-user support, enhanced HTML export facilities and the automatic path generation facility. HyperTech 3.0 was a port of the previous work to the Solaris 2.5.1 platform and was delivered

in August 1997. Along with a number of usability and performance enhancements, it contained a facility for importing HTML data from the World Wide Web (WWW) via a remotely controlled Netscape Navigator™ browser.

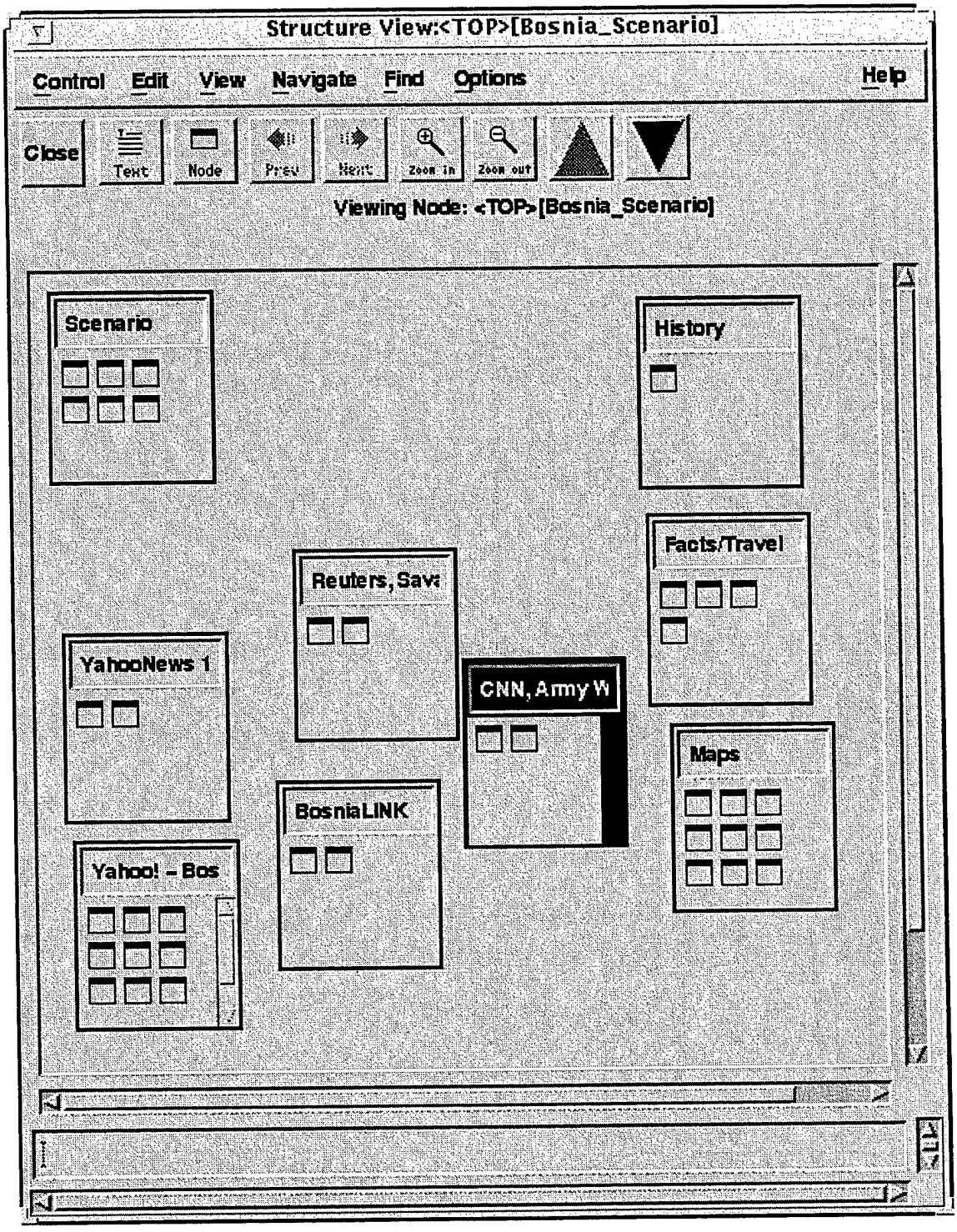
1.2 Compatibility with HyperTech 2.0

The primary goal of the HyperTech 3.0 effort was to port the 2.0 functionality to the Solaris platform. Accordingly, the points made in the earlier Final Technical Report for HyperTech 2.0 are all relevant to this final release. For convenience, we have included a copy of the earlier Technical Report as Appendix A, and will periodically refer to this document for additional background. References will be made in-line with square brackets containing the abbreviation **FTR2**, followed by the section number. For example, [**FTR2:**2.1] refers to section 2.1 of the earlier report.

1.3 HyperTech 3.0 Functionality

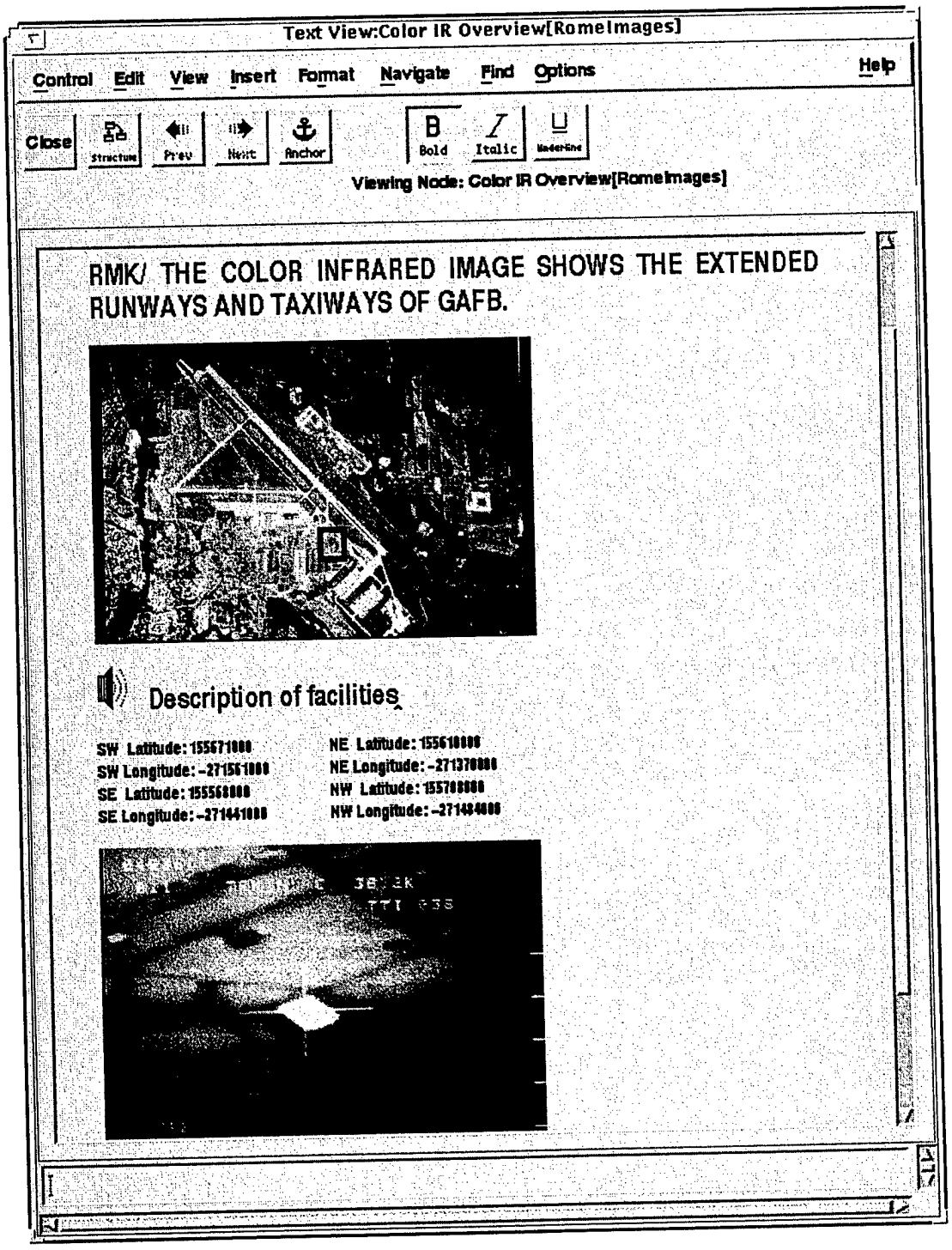
The following pages in this section illustrate the major views and interfaces offered by the HyperTech environment.

Structure View



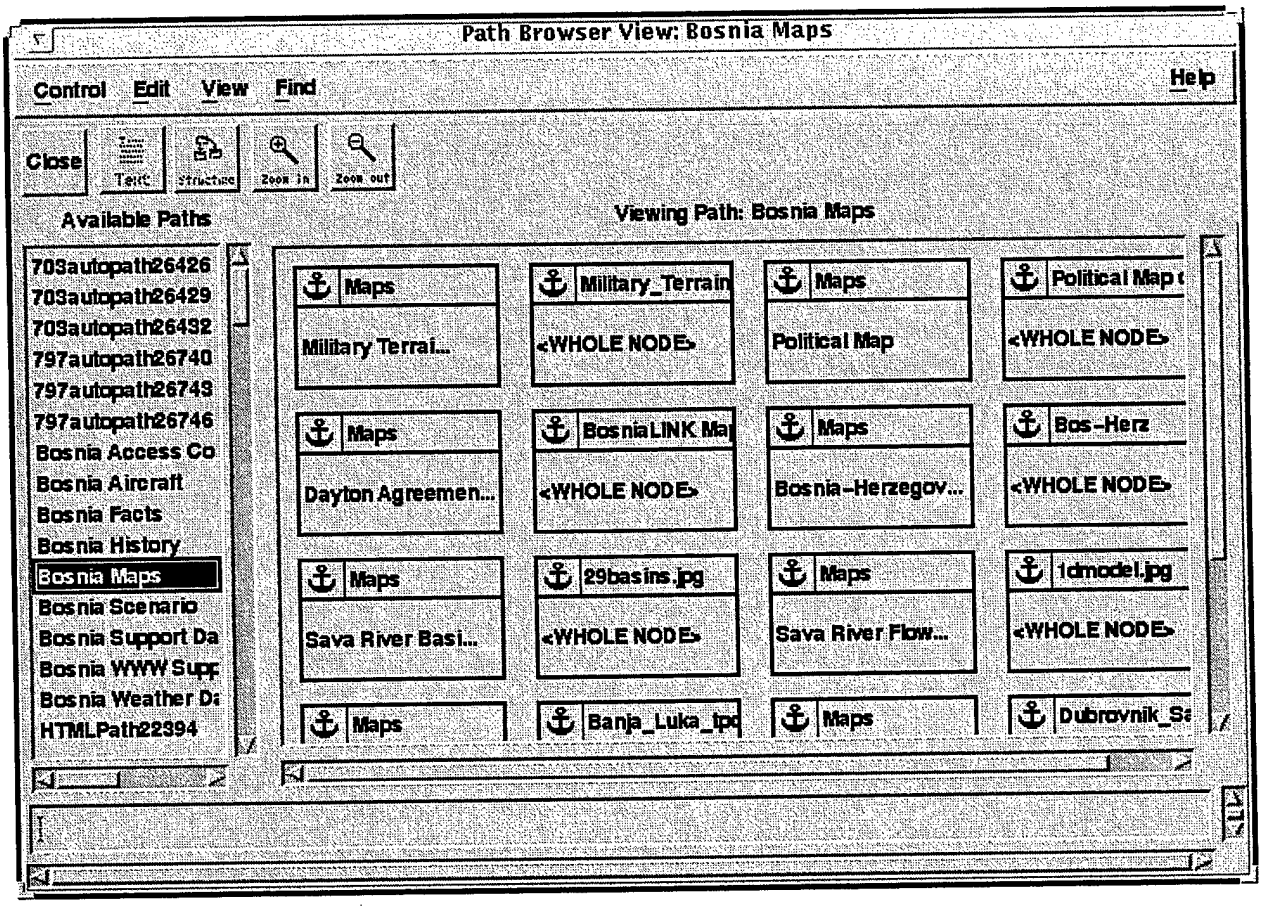
The Structure View represents information nodes as boxes. The smaller boxes within these top-level nodes are child nodes. These nodes may also have children, etc. The arrow buttons can be used to move about the node hierarchy. The Structure View allows information to be organized, accessed, and manipulated with the drag-and-drop user interface. A double-click on a node will open that node's text view.

Text View



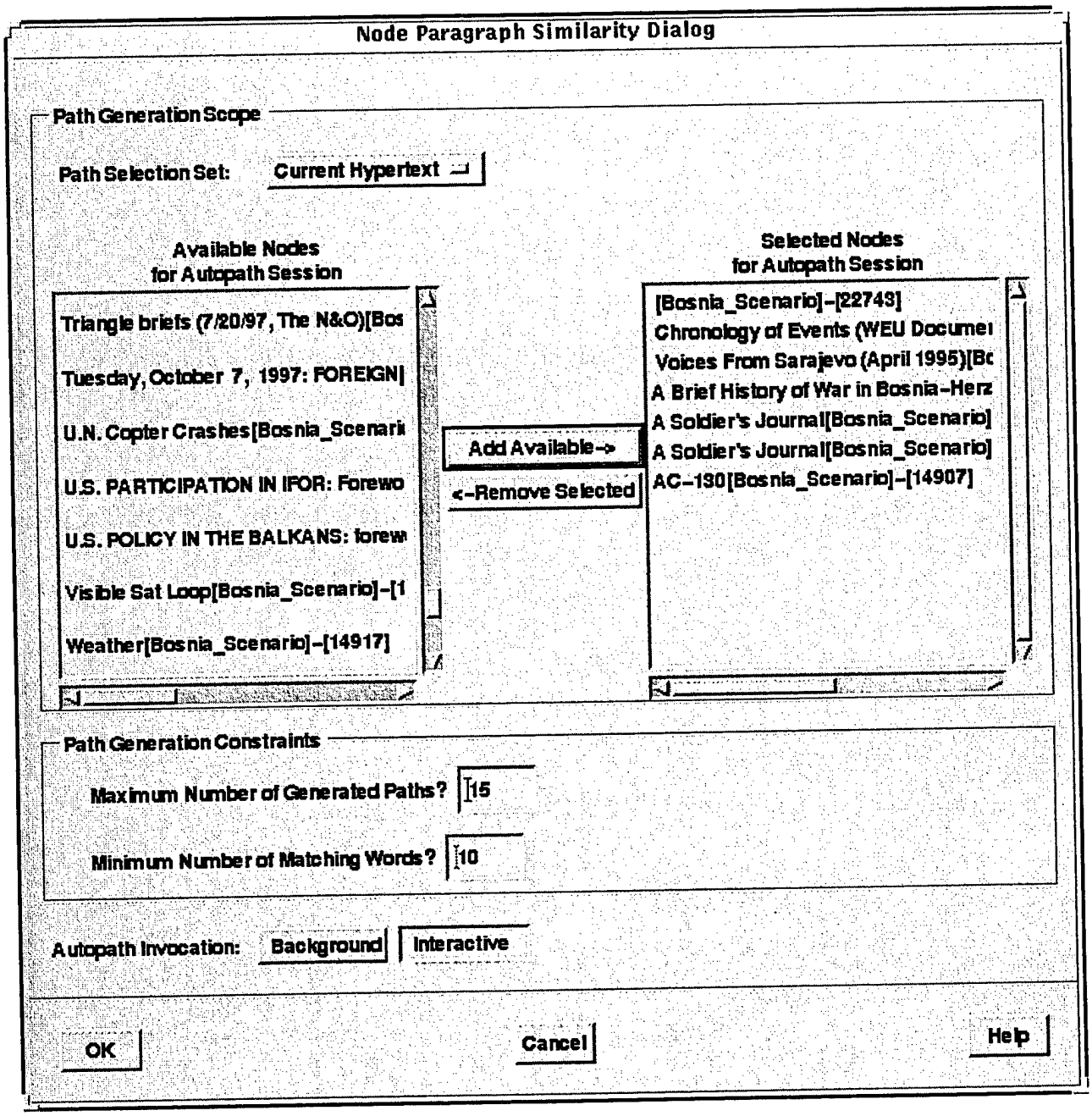
The Text View displays the data contained within a node. It may contain graphics, video, audio, overlays, and stylized text. Red text indicates that text is an anchor on a path. A graphic or an entire node may also be an anchor. The directory location of any non-text data can be revealed. A helper application appropriate for non-text data (e.g., XV for images) can be initiated from this view. Formats supported include graphics (GIF, TIFF, PNM, JPEG), video (MPEG-1), audio (AU), and text (ASCII).

Path Browser



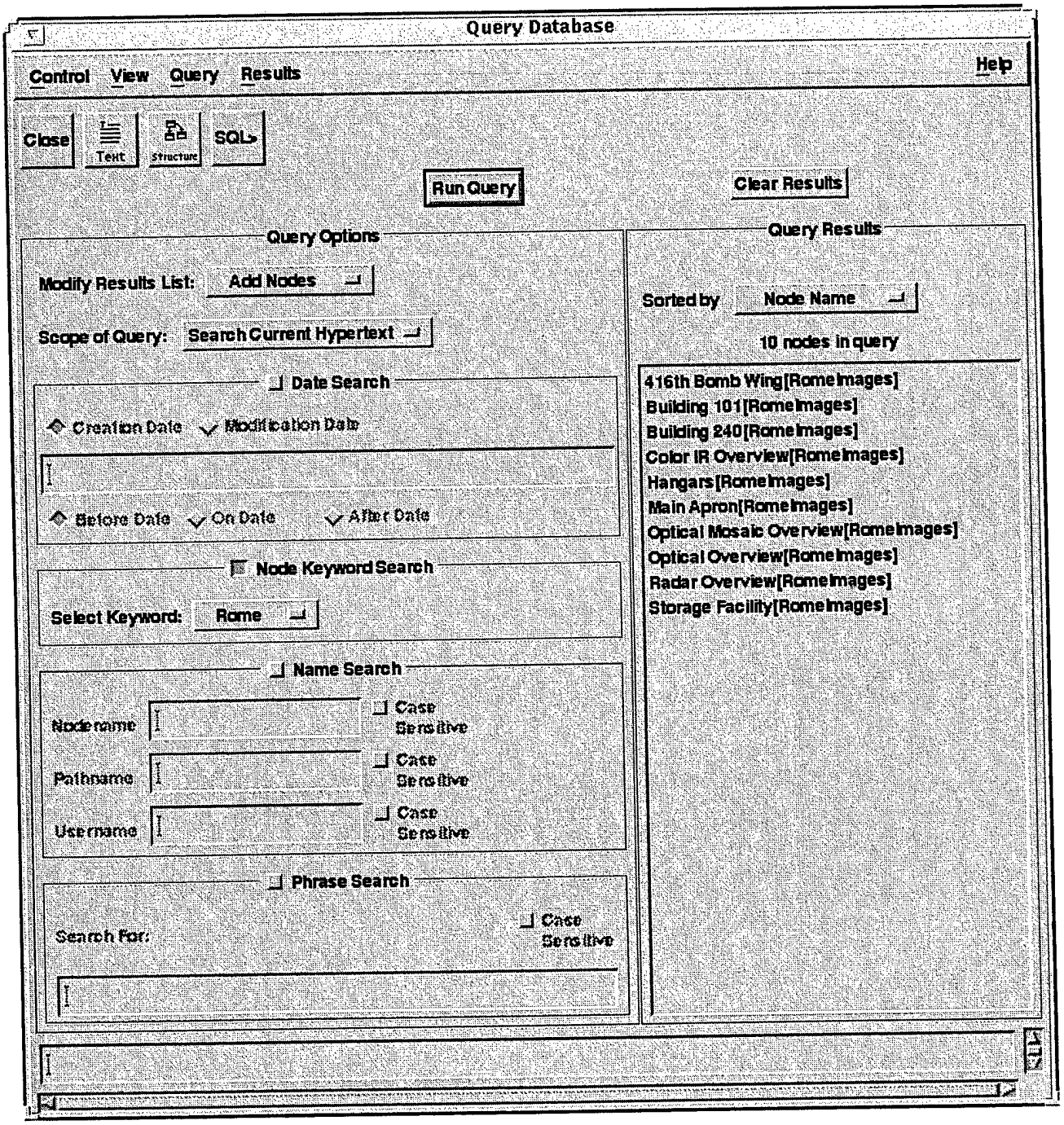
Two or more anchors form a link, and a path is formed by adding additional anchors. The Path Browser displays anchors represented as boxes. The title of each box is the name of the node containing the anchor. Inside the box is text describing the nature of the anchor (a piece of text, a whole node, or a graphic). Anchors may be dragged from Text and Structure Views into the Path Browser. Anchors can be rearranged within the Path Browser.

AutoPath



The automatic path generation facility (AutoPath) automatically creates paths between related components of a HyperTech database. Four AutoPath options exist: Node Similarity, Paragraph Similarity, Node-Paragraph Similarity (dialog box shown above), and Text Match Similarity. Parameters may be specified. AutoPath may be run in the background or in interactive mode. Results appear in an AutoPath Browser similar in appearance and function to the Path Browser.

Database Query

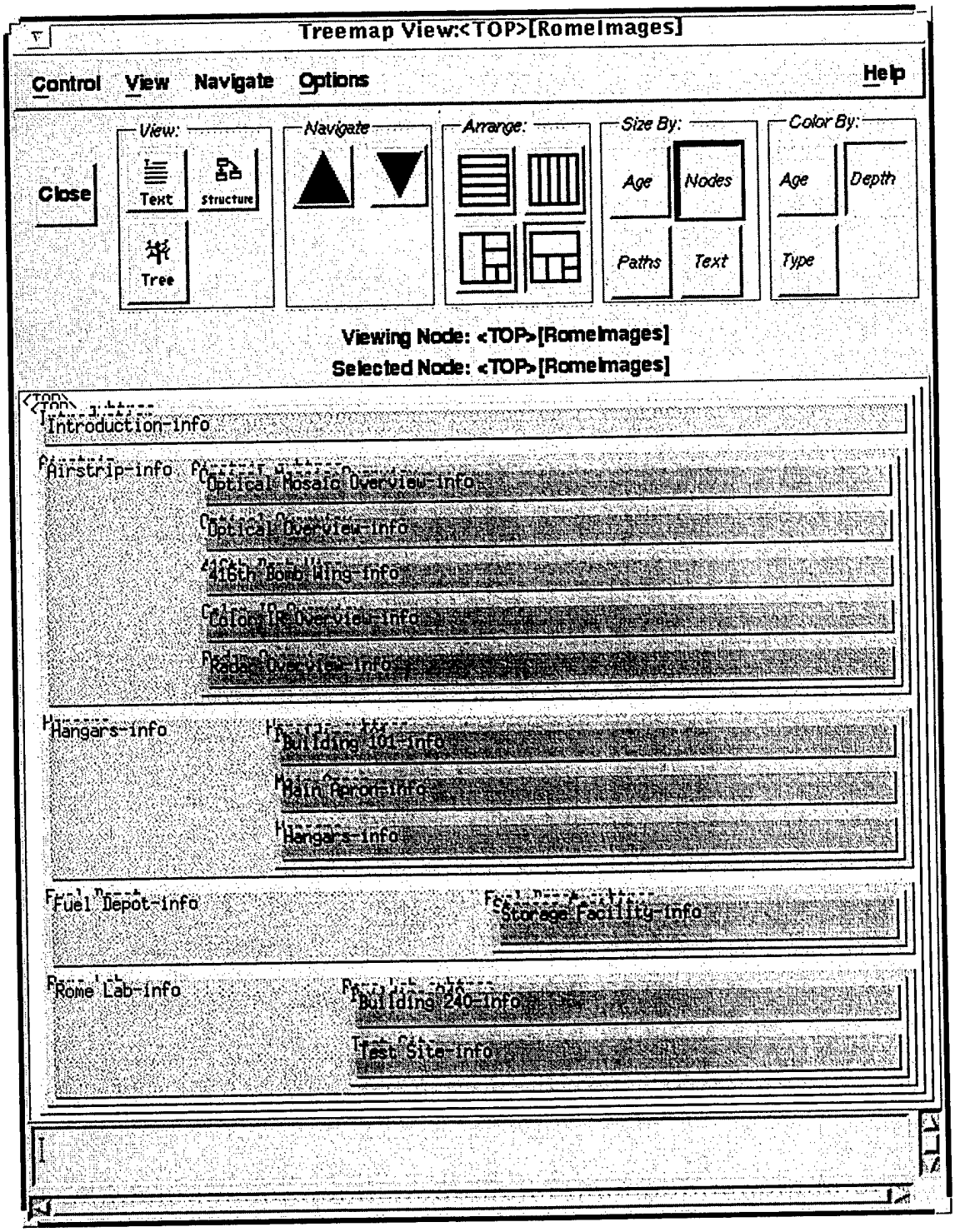


Four types of searches may be made from the Database Query window:

Name Searches – to find nodes with a particular name.
Phrase Searches – to find nodes which contain a text phrase.
Date Searches – to find nodes which were created or modified before, on, or after a date.
Keyword searches – to find node that have been labeled with a keyword.

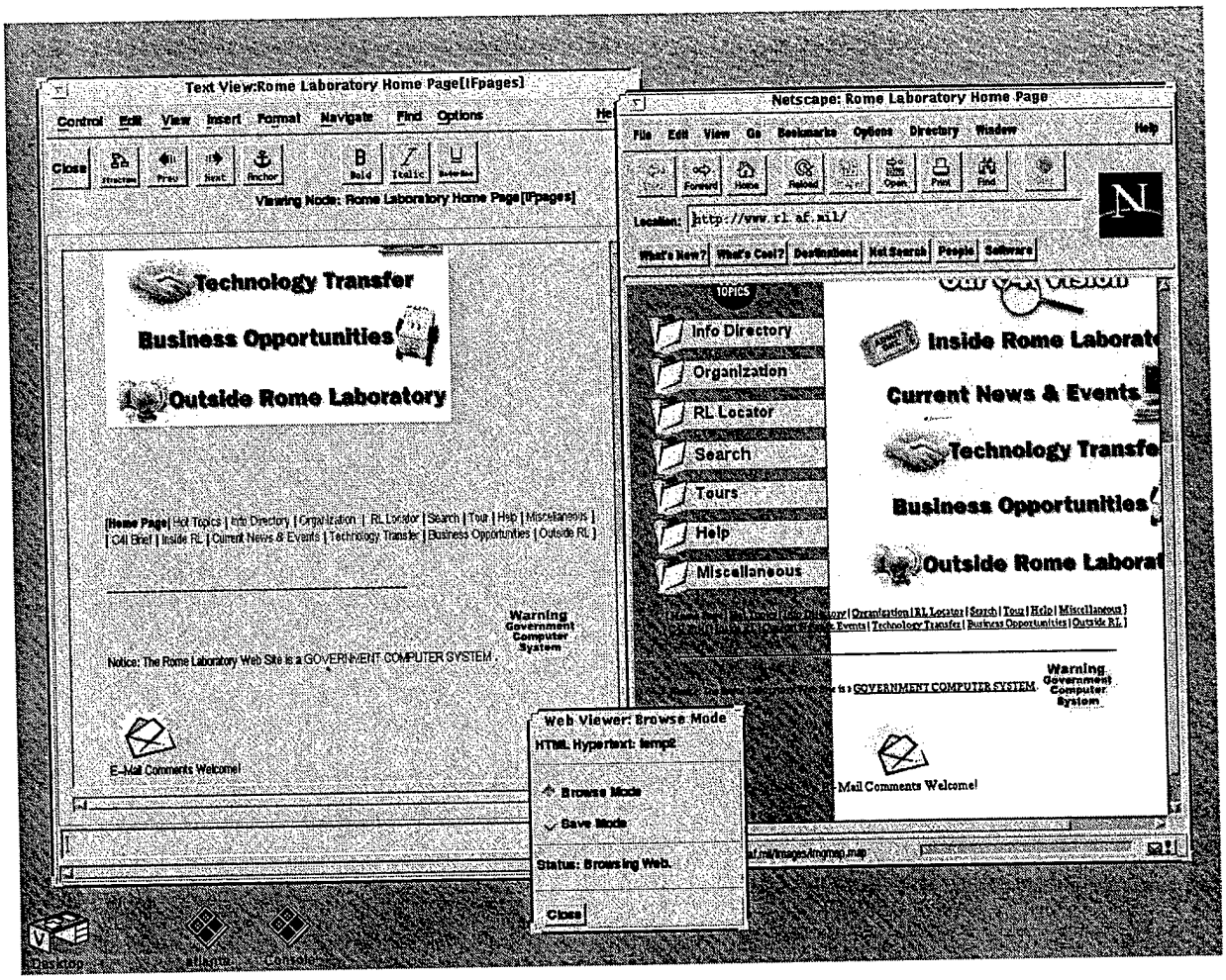
Nodes resulting from queries can be automatically formed into a path.

Treemap View



The Treemap View provides an overview of a hypertext. Each rectangle represents a node. The amount of text content within a node determines the size of the rectangle. Children nodes are contained within their parent node. Options also exist for rectangle size to represent number of children, node age, or number of paths passing through a node.

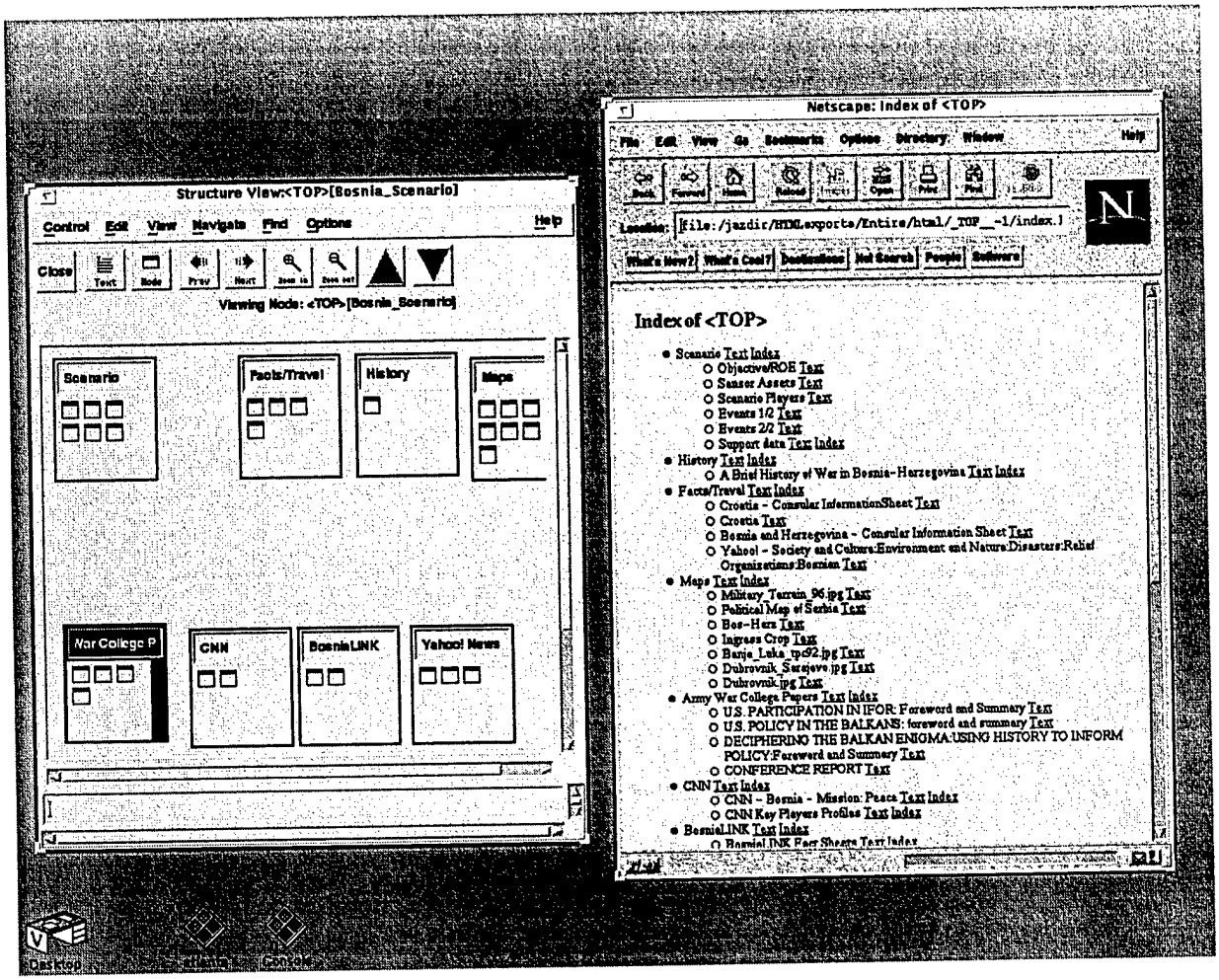
WWW Import



Shown above: HTML displayed in Netscape Navigator 3.0™ browser on the right, Web Import browse/save toggle dialog at middle bottom, HyperTech representation of imported HTML on the left.

HyperTech provides HTML import support by using a Netscape™ browser remotely controlled by the HyperTech process. Netscape™ provides the HTML content for the main page, while a special client library is used to retrieve images and audio files referred to within the URL (see section 2.12).

HTML Export



Shown above: HyperTech structure view on the left, with its corollary HTML index shown in Netscape Navigator 3.0™ browser on the right.

HyperTech has a HTML export capability, allowing the user to take information stored in a HyperTech database and convert it to the closest HTML equivalent. Standard HTML tags and attributes cannot easily represent the HyperTech path facility, with its concept of an ordered traversal through a set of anchors. Nor can standard HTML represent the ability to have multiple connections through the same anchor. Work-arounds for these conditions have been implemented (see section 2.12).

1.4 Reference Documents

Documents pertinent to, or generated as a result of this project are listed below, along with brief descriptions of each.

- *HyperTech Users Guide Version 3.0*, GTRI Project Number A-5069, Georgia Institute of Technology, 22 August, 1997. Users Manual accompanying Version 3.0 of the HyperTech software, delivered 25 Aug 1997.

- *Evaluation of HyperTech using Think-aloud Protocols*, GTRI Project Number A-5069, Georgia Institute of Technology, 5 May, 1997. Usability test procedures and results.

- *Hypermedia Interface Interim Prototype Final Technical Report*, RL Technical Report Number RL-TR-96-202, Georgia Institute of Technology, Dec 1996. Final report on the first phase of hypermedia development (HyperTech 2.0).

- *System/Segment Design Document for the Hypermedia Interface Interim Prototype,* GTRI Project Number A-9575, Georgia Institute of Technology, 31 March, 1994. Initial system design document for the program. Subsequent design changes necessitated major revisions to this document.

- *Software Test Plan for the Hypermedia Interface Interim Prototype*, GTRI Project Number A-9575, Georgia Institute of Technology, 16 May, 1994. Document describing testing procedures to be used for developed software.

- *Software Design Document for the Hypermedia Interface Interim Prototype*, GTRI Project Number A-9575, Georgia Institute of Technology, August 1996. Final version of the V2.0 HyperTech System Design, documenting the software design as delivered May 1996.

- *Architectures for Volatile Hypertext*, Bernstein, Bolter et al., Hypertext 1991, New York: ACM, pp. 243-261. Article describing hypertext applicability to information management, written by one of the project's principal investigators.

- *Writing Space; The Computer, Hypertext, and the History of Writing*, Bolter, Jay David, Lawrence Erlbaum Associates, Publishers, 1991. The study of the computer as a new technology for reading and writing.

2. Detailed Discussion

2.1 Multi-user Support

The initial System Design Requirements for HyperTech did not address support for concurrent access to a HyperTech database. The first database design was geared towards a single user with complete control over a specific database, although multi-user capabilities were envisioned for some point in the future.

Per the original requirements, HyperTech 1.0 was delivered as a single-user system. As the scope of the application grew, it became increasingly clear that multiple users would need to be able to read data from a common database. Locking out HyperTech users who wished to browse a hypertext stored in the same database as a hypertext being modified seemed unnecessarily restrictive. With the addition of the Automatic Path Generation batch facility, which required that two users be able to access the same database simultaneously (the human user as well as the Auto-Path batch process), a redesign of the database schema for better support of concurrent users was clearly needed.

The redesign was complicated by Sybase's concurrency control mechanism. While some other database engines provide record level locking, the Sybase server does not. When a record is being modified to reflect changes to its associated HyperTech object, Sybase restricts access to all records within the same 2K data page. Records which have nothing to do with the modification are locked, simply because they are in the proximity of an affected record.

The final solution, delivered in HyperTech 2.0, supports multiple readers and a maximum of a single writer in a specified database. The user may commit or revert changes in a session at his or her discretion, just as with the single-user design. At the same time, we maintain data consistency and do not permit readers to access uncommitted data in use by an active writer. Additional details on the design rationale appear in **[FTR2:**2.1].

In HyperTech 3.0 some minor bug fixes were added to correct errors where the data model failed to take versioning into account when operating on the database. Beyond these changes, no other modifications were made to this functionality for HyperTech 3.0.

2.2 Use of Xew Widget Set

Finding an existing Motif widget to support the display of multimedia data and provide WYSIWYG editing features proved to be a challenge. Our goal was to use an existing widget, rather than develop one of our own. The widget needed to support styled text, in a variety of fonts and sizes, along with support for embedding graphics, video and audio data. It also needed to be "Motif-aware". The Motif Text Widget provides none of these features, so we initiated a search for public domain widgets that would provide these features.

After examining a number of possible candidate widgets, none of which satisfied all of the above requirements, we settled on the Text Editor Widget in the Eurobridge Widget Set, commonly referred to as Xew. Xew contains a collection of widgets which were used as the basis for

HyperTech's Text View. The only requirement it did not meet was integration with the Motif widget hierarchy. As part of the HyperTech project, we rectified this flaw by adding the necessary code to make the Xew widgets Motif-aware, which we submitted to the maintainer of the Xew toolkit in early 1995.

As noted in [**FTR2:**2.2], the Xew widget set suffers from some inherent problems in terms of functionality and performance. Nonetheless, it was the best choice at the time that the bulk of the implementation was written — the alternative of implementing our own widget set would have been a tremendous undertaking and would have take resources away from other aspects of HyperTech development.

We had hoped that the final version of the Xew widget set would address some of our concerns. However, in production of the final release of Xew, its maintainer rewrote some of the internal code upon which our contributed modifications were based. This caused the GTRI-contributed code to no longer work correctly. It would have been necessary to recode and retest all of our previous modifications in order to restore the support which was broken in the new release.

In addition, the final Xew release contained new bugs and did not address our specific problems with performance or lack of customization. Therefore the final HyperTech delivery was based upon an older (but for our purposes more stable) version of Xew.

Other text editor alternatives were considered during this phase of the HyperTech project. Having previously used Xew, we had exercised maximum efficiency by storing textual data in Xew's native storage format. Knowledge of this format, a subset of the ISO 6429 standard for controlling text layout and appearance, was encapsulated into a separate module in the data model. Both the data model and the user interface utilized this module for manipulating textual data. Therefore, a new toolkit implementation would have required not only a redesign of the Text View, but would also have forced the issue of a more appropriate storage format, one natively understood by the new Text Widget. As discussed in Section 2.3, an HTML-based storage format is felt to be most appropriate. That storage requirement logically dictates the need for an HTML editor widget to implement the Text View. These two changes would have necessitated a radical rewrite of key components of HyperTech, a rewrite unfeasible for the scope of the third and final phase of the HyperTech project.

2.3 HyperTech Storage Format

As mentioned in Section 2.2, HyperTech stores its text data in the native storage format used by the Xew Text Widget. In retrospect, it would have been better to store the data in the Hyper Text Markup Language (HTML) format. Direct support for HTML tags would have made it easier to export and import HyperTech data into a format suitable for the World Wide Web, functionality which was later added to the HyperTech system in versions 2.0 and 3.0 respectively. However, the immaturity of the World Wide Web and the HTML standard circa 1994, along with the restrictions of the early version of the Xew Text Widget, made the Xew storage format the best choice at the time. The full discussion of this decision is in [**FTR2:**2.3].

Any future work with an HTML storage format will be able to leverage the existing HTML import and export modules. This support would be the basis for developing a converter from the existing HyperTech data format to an HTML-based format.

2.4 Enhancement of Navigation Metaphor

The navigation metaphor implemented for HyperTech 2.0 has continued to work well for HyperTech 3.0. Creating anchors (marked ranges of text or locations in an image), combining those anchors into paths (dragging and dropping on-screen anchor representations into a path builder), and navigating the path (clicking on a specific anchor and navigating the path with left and right pointing arrows) have all proven to be easy and natural for the user. No changes to the metaphor were made during this phase of the project.

2.5 Data Acquisition

As in earlier phases of the project, we were unable to obtain real world or simulation data for manipulation with HyperTech. AFSOC was unable to provide a data set. Other government programs (eXtended Integrated Data Base [XIDB], etc.) were surveyed, but no suitable unclassified data repository was surfaced.

Our work-around was to develop the HTML import module, which allowed incorporation of multimedia data from the World Wide Web into HyperTech. This proved to be a reasonable mechanism for obtaining large amounts of data containing integrated text and images.

2.6 Automatic Path Generation

The automatic path generation facility remained unchanged during this phase of the project. Improvements to its user interface were made, but the AutoPath facility itself was not modified.

2.7 User Interface Techniques and Usability Testing

Usability testing on the HyperTech interface was conducted during this phase of the project. The stated goal of the testing was to evaluate some of the hypermedia features within HyperTech which differ from hypertext as it appears on the World Wide Web.

The evaluation of HyperTech, led by our Graduate Research Assistant Teresa Hubscher-Younger, was a "think-aloud" protocol analysis. In this type of test, a test subject uses the system to complete a variety of tasks, and thinks aloud, constantly verbalizing what he or she is planning, doing, and perceiving as the task proceeds.

Specifically the tests were designed to cover the following functionality:

- –multiple paths through the same anchor
- –multiple navigation methods
- –multiple search methods
- –automatic path generation

After analyzing the test results, the testing team felt that future tests would need to include a training element on the rationale and concepts of the advanced functionality being tested before attempting to test the implementation of the concepts. While users were very familiar with the World Wide Web's concept of hypertext, the more sophisticated functionalities in HyperTech, which lack a WWW analog, required additional explanation. This made it difficult to complete the testing tasks in the allotted time. In this sense, the tests were inconclusive — more time was spent explaining the concepts rather than testing the implementation of the concepts.

However, the team was able to provide recommendations for improvements to the HyperTech user interface. Based on these suggestions, a number of modifications were incorporated into the HyperTech 3.0 user interface. Details on the recommendations and steps taken to address the concerns appear in Appendix B of this document.

Although the testing team did not directly accomplish the original goals, the incidental testing feedback provided was invaluable. The feedback from the subjects, in addition to suggestions from the project's Rome Laboratory program manager, made significant contributions toward improving the interface of the final product. Testing by AFSOC personnel was not available, as their time available to contribute to the effort was limited.

2.8 Design of User Interface and Data Model

HyperTech was designed as a set of three software layers to facilitate potential future ports to different database engines and user interfaces. Those layers are:

- The Data Model Access Layer (database independent);
- The Data Model Implementation Layer (database dependent);
- The User Interface Layer.

Full details on these different layers appear in [**FTR2**:2.8]

No changes were made to this approach for HyperTech 3.0. It continued to be a clean way to delineate boundaries between the user interface, a general database programming interface, and the database engine-specific library used to implement the general database API.

For future work, the front-end/back-end approach to the data model is analogous to the Java Database Connectivity API (JDBC). JDBC also provides a database-independent API and a database-dependent implementation of the API. The similarity of the approaches should simplify the mapping of HyperTech concepts onto a JDBC implementation in the event of a Java-based HyperTech development.

2.9 Object-Oriented Software Development

As discussed in [**FTR2**:2.9], all phases of HyperTech were implemented using an object-oriented approach to writing an ANSI C application. While appropriate for development of the existing product, future work should be done in a true object-oriented language to utilize the inherent support for designing reusable and encapsulated components, rather than continue to graft such support onto an older language like C.

While the C++ language continues to be popular, the use of the Java programming language would be a more appropriate language choice for future development. As of this writing, Java is becoming the preeminent development platform for sophisticated cross-platform Web-based applications. To recast HyperTech functionality into a new Web-based version will almost certainly require a Java based implementation. Java was still immature at the onset of this effort, and C met the needs of development of the prototype.

2.10 Performance Issues

Porting HyperTech from SunOS to Solaris resulted in overall improved performance. The Solaris OS improved performance of many underlying system calls, and the same version of the Sybase database engine under Solaris ran significantly faster than its SunOS counterpart. The newest version of Sybase (Version 11.0) was released midway through the project and also provided a significant performance increase.

Throughout the final phase of HyperTech, we analyzed the application with the *Quantify* performance analysis tool. Problems in displaying shallow hierarchies (mentioned in [**FTR2**:2.10]) were addressed, along with a number of other subtle performance bottlenecks in the search and retrieval of the database. Performance deficiencies in the Text Widget remained, even on the new platform. Overall, HyperTech 3.0 runs much faster than HyperTech 2.0, although operations like converting to and from HTML still take significant time. The latter would be resolved by storage in HTML, eliminating the need for a conversion step.

Future efforts would benefit from a rewrite of underlying data model components. The data model still shows some vestiges of its original design for a single user in a single session. In the process of re-implementing with an object-oriented language, there will be opportunity to redesign the application to develop more efficient implementations of key functionality.

2.11 Port to Solaris

Porting the HyperTech application to the Solaris OS proved very straightforward. We attribute this to the use of POSIX standard system calls, isolation of platform-specific code, and a layered approach to the major modules in the system. As mentioned in Section 2.10, some performance enhancements were achieved simply by moving to the new platform due to improvements in the underlying operating system and supporting tools.

2.12 HTML Export and Import

HyperTech 2.0 contained an HTML export facility, allowing the user to take the information stored in a HyperTech database and convert it to the closest HTML equivalent. Standard HTML tags and attributes cannot easily represent the HyperTech path facility, with its concept of an ordered traversal through a set of anchors. Nor can standard HTML represent the ability to have multiple connections through the same anchor.

To compensate for the lack of HTML functionality, we developed a mechanism for representing paths in standard HTML to provide the equivalent information. Anchors which were used

multiple times are represented by a single anchor reference. Clicking on the anchor reference brings the browser focus to an HTML list of all possible locations and paths which utilized the corresponding HyperTech anchor. A special back anchor or anchors was generated around each HTML destination anchor to indicate the URL from which a reference was made. Hierarchy was preserved by generation of a special page of HTML showing all the children of a given node, separate from the contents of the node.

HyperTech 3.0 provided HTML import support, by using a Netscape™ 3.0 browser (the prevalent browser at development time) remotely controlled by the HyperTech process. Rather than develop yet another web client, we wished to leverage familiarity with an existing browser. Netscape™ provides the HTML content for the main page, while a special client library is used to retrieve images and audio files referred to within the URL.

By default, the import facility is a passive observer of the browsing session. To allow selective import of Web pages or URLs into the HyperTech system, the data is imported only when the user has specifically toggled the HTML Import facility from Browse mode to Save mode. Thus the user can browse until he or she finds the page that needs to be imported, toggle HyperTech into Save mode, capture the contents of the URL, and then switch HyperTech back into Browse mode. This allows the user to avoid saving pages which are traversed merely as stepping stones to the desired page. Because standard HTML contains a number of tags which do not map onto the HyperTech data storage format (for example, <TABLE> and <FORM>), the import process will convert those tags which map naturally onto its available formats (BOLD, ITALIC, list tags, etc.), and ignore the others, in accordance with standard browser practice of ignoring tags which it cannot render.

Finally the HyperTech organizing metaphor is applied to the imported data. Hierarchy is generated by creating parent-child relationships between nodes containing a reference to an anchor, and nodes which contain the anchor, taking care to avoid circular hierarchies. Two special traversal paths are generated for each browsing session. The traversal path is a path linking in order each page that was visited in the browser and saved into HyperTech. Every time a page is visited in the browser, an entry is made into the traversal path. The capture path is an optimized version of the traversal path, showing all the pages imported into HyperTech in the order they were captured, but eliminating revisits of the same node.

While this approach to importing HTML has met the stated goal of providing access to the World Wide Web within HyperTech, it suffers from some shortcomings due to deficiencies in the Netscape™ remote control protocol. For example, Netscape™ does not notify monitoring processes when it has finished retrieving a URL, forcing us to come up with alternate ways of determining when the URL has been fully downloaded. Also, the protocol only works for Netscape™ browsers, so the user cannot use a different browser in conjunction with HyperTech.

The protocol is also inconsistent between different versions of Netscape™. The developers of the latest version of Netscape™, Communicator 4.0, inadvertently broke some features of the remote control mechanism, and although they plan to restore the mechanism, it is not yet clear when that will happen. Our investigation into writing a Netscape™ plug-in, in hopes of circumventing this last problem, were unsuccessful — the information required is not available

via the plug-in API. Nonetheless, use of Netscape™ 3.0 was sufficient to prove the concept of the prototype.

Future versions of HyperTech should understand HTML natively and be more directly integrated into the Web (most likely through an HTML-based text editor as discussed earlier). In this way, the issue of having to import or export HTML becomes irrelevant — by using HTML as the native format, no conversion is necessary. With better Web integration, the need to rely on a sometimes opaque process for information on the URL is obviated.

Netscape™ is a trademark of Netscape Communications Corporation.

3. Areas for Potential Future Development

3.1 Enhanced Multi-user Support Features

The versioning approach used to implement multi-user support in HyperTech offers a number of possible enhancements. Automatic notification of newly published versions of HyperTech databases to existing readers could be provided via the addition of a simple protocol. This automatic notification could allow users the option of examining the latest version of HyperTech data when they become available. Automatic deletion of older versions (after the last reader of the old version exits) could be incorporated into HyperTech. We could also examine the feasibility of reducing the scope of the write lock, from a databases perspective down to a hypertext, or even a section of a hypertext, if the underlying database supports such fine grained locking.

3.2 User Interface Toolkit Alternatives

As discussed in Section 2.2, future work will necessitate replacement of the underlying toolkit used to implement the multimodal Text View. Any such toolkit will need to work with HTML data and will most likely need to provide a Java API. Because of the multi-layered approach to the HyperTech interface, this will involve developing a new interface to interact with the data model access layer.

The new interface would be likely to include some combination of Java Bean components. A Java Bean is a reusable lightweight software component, similar to a Microsoft Active X control on the MS Windows platforms. Beans are Java objects which perform a specific purpose ranging from a small email reminder all the way up to a full scale HTML text editor. Beans are written according to a standard specification for inspecting and customizing Bean properties and for "publishing" a list of callable methods. Adherence to this standard makes it possible for a graphical application builder to glue different Beans together into an application without any integrating code.

New standards for combining Beans with other Beans are under development. This will allow Beans to contain other Beans and to exchange information according to standard Bean interfaces. A Text View could thus be implemented as a set of off the shelf Beans, e.g., as a component Text Editor Bean which also communicates with special Audio and Video Playback Beans. Adding support for a new type of data or data format might be as easy as connecting a new Bean into the existing framework.

3.3 Data Storage Alternatives

As discussed in Section 2.3, we would recommend future versions of HyperTech store text data in an HTML-style format. Such a modification would simplify the process of importing and exporting data from the World Wide Web. To represent functionality that is not present in standard HTML (the path navigation mechanism) it would be reasonable to add a new attribute or attributes to anchor tags to encode this different information. Browsers which are not capable of these HTML constructs will simply ignore the extra attribute(s), which means that the HTML used to represent HyperTech data could trivially be exported to a Web browser of choice.

An effort that could impact future HyperTech efforts is the development of the Extensible Markup Language (XML) by the World Wide Web Consortium (W3C). The XML standard, currently early in the working draft stage, is a simplified set of conventions for using Standardized General Markup Language (SGML). SGML is a large and complex set of conventions for combining multiple kinds of data into a user-defined document structure, referred to as document type definitions or DTD. HTML is an example of SGML-compliant grammar and processing conventions. SGML is large and unwieldy, and part of the rationale for the development of XML is to (quoting from the XML FAQ) "enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML".

Of special interest to our work is a subsection of XML which will designate standards for creating more powerful linking capabilities than are available in the current HTML standard. This would include bi-directional links and multi-ended links, such as provided by HyperTech today, along with additional capabilities such as link typing and link descriptors. Should XML become a widely adopted standard, it would make it even easier to represent HyperTech concepts in a common format.

3.4 User Interface Enhancements

For browsing purposes, we believe it would be useful to have a view which serves as a combination of the Structure View and the Text View. This view should show both the hierarchy contained within a node and the text of selected nodes within that hierarchy. Such information is currently available by launching a Text View from a selected node in the Structure View; but once the Text View is launched, it is decoupled from its Structure View. The combination view would dynamically update the text displays as different nodes are selected in the hierarchical display. This view would also address concerns about the proliferation of windows launched within HyperTech.

3.5 Porting to the Java Language

We believe that a port of HyperTech to the Java language would be extremely valuable in at least three areas. First, a port to a true object-oriented language would facilitate the application of object-oriented techniques to HyperTech development. A true inheritance hierarchy for the views could be provided, and message passing schemes could be more easily and rigorously enforced.

Second, a Java-based HyperTech could work with popular Web browsers such as Netscape™ or Internet Explorer, and provide access to HyperTech data within the framework of a well known application. Integrating HyperTech with a Web browser would make HyperTech more accessible to the Web, and the Web more accessible to HyperTech. HyperTech could be designed as a set of applets working within the browser framework, or within its own Java application framework interchangeably.

Finally, such an implementation would be platform-independent, allowing HyperTech browsing and editing on any Windows, Macintosh, or UNIX system with a network connection to a database server. In fact, use of Java Database Connectivity within the data model could make

HyperTech database-independent as well, without needing to write any database-specific code. If the database-dependent portion of HyperTech was implemented in terms of the JDBC API, then it would immediately support all database engines which provide JDBC API implementations.

3.6 Performance Enhancements

Significant performance improvements were made between HyperTech 2.0 and HyperTech 3.0. A future revamping of the data model will definitely improve access to the underlying database.

Throughout this document, we have stressed the need to develop a Java-based implementation of HyperTech. It is important to point out that the way Java code remains platform-independent is by compiling human-readable source into an assembler-like output called "bytecode". Bytecode is interpreted at runtime by the Java Virtual Machine, which contains all the platform-specific code for executing the application on a specific machine. The cost of maintaining platform-independent code is discovered at runtime, because the interpreted code runs more slowly than machine-specific code.

New variants of Java compilers are coming on the market to address these performance concerns. Bytecode optimizing compilers streamline the output of a standard Java compiler into more efficient bytecode sequences. "Just in-Time"(JIT) compilers dynamically compile designated bytecode entering the Java Virtual Machine into native machine code, preserving the platform independence of the Java bytecode while providing some of the benefits of native code. Hybrid compilers can generate combinations of Java bytecode, shared libraries, or native executables depending on user directives. As the Java market grows, so will the emphasis on producing fast machine-independent code, to the point where Java Virtual Machines will run Java applications as quickly as many native C applications, with the added benefit of security and integrity not enforced in the C language.

4. Conclusions

Our experiences with the design and development of the HyperTech prototype have been encouraging. We feel that the user interface simplifies the understanding of hypertextual data and provides reasonable facilities for editing and traversing their components. The automatic path generation facilities are of tremendous value in reducing the burden of analyzing texts and creating connections between related concepts.

We also believe that HyperTech currently provides features that no other hypertext system does, namely:

- storage of hypertext data in a relational database that is manipulated in an object-oriented, drag-and-drop manner.
- support for organizing a graphical hierarchy in a hypertext, in addition to traditional non-linear connections; it is felt that this provides greater locational awareness than current access techniques on the World-Wide Web – lessening the chances of becoming "lost in hyperspace";
- multiple, alternative views of the data space to aid information finding and navigation;
- import and export facilities to transfer data in a variety of formats, including HTML;
- support for multi-user access and read-only and write modes – 1 writer, many readers at any one time;
- automatic path generation based upon text content for automatically building links within hypertexts, or discovering relevant information;
- the data base foundation of HyperTech provides access controls, data consistency, and a large data capacity.

HyperTech is a flexible hypermedia environment which allows users to integrate, visualize, and manipulate large collections of multimedia data (text, imagery, video, audio, overlays). HyperTech permits multimedia data to be organized and linked so that the information analyst can quickly exploit these relationships.

HyperTech can be used as an automated tool to help build hypertexts without HTML coding skills. The HTML export feature then makes these hypertexts accessible to a large audience by publishing in a format readily viewable by popular web browsers. The HTML import feature allows the user to import information available on the Internet (or Intelink, etc.). HyperTech can then be used to organize and integrate this information with any existing information. Any portion of the subsequent information web can then be exported in HyperTech or HTML format for dissemination. HyperTech, itself, can be used as a tool to organize and present flexible briefings.

In the year and a half since the release of HyperTech 2.0, the need to integrate HyperTech functionality into a WWW/Java-based environment has become central to any future efforts. A Java solution would make the technology developed more widely available and easily integrated into existing desktops. Acceptance of HyperTech has been hampered by the fact it is a standalone application. By making HyperTech work with the defacto standard for hypertext data (HTML) and by making it more easily integrated into the daily workspace of its candidate users (a Web browser environment) would require a much smaller investment on the user's part — an

interested party can simply install it on the desktop or run the Java component on a Web server, and immediately test it with their existing HTML data. After users gain some experience with HyperTech, we believe that they will find its extra features beneficial for manipulating and visualizing hypermedia data.

By making HyperTech part of the everyday environment, users will have more incentive to utilize its advanced functionality, and to store their multimedia data in the organizing structures it provides. We feel this is the best opportunity for HyperTech to achieve its potential and to become an important tool in the creation and analysis of complex multimedia data.

APPENDIX A

Selected Sections from the Final Technical Report
for
Hypermedia Interface Interim Prototype
(HyperTech 2.0)

published in
Rome Laboratory Technical Report #RL-TR-96-202
December 1996

Georgia Tech Research Institute
Ms Susan Liebeskind, Dr Jay D. Bolter, Mr Phillip Hutto,
and Mr Kirk Pennywitt

Section 2: Detailed Discussion
Section 3: Suggestions For Future Work

Rome Laboratory
Air Force Material Command
Rome, New York

2. Detailed Discussion

2.1 Multi-user Support

The initial System Design Requirements for HyperTech did not address support for concurrent access to a HyperTech database. The first database design was geared towards a single user with complete control over a specific database, although we envisioned adding multi-user capabilities at some point in the future.

Per the original requirements, HyperTech 1.0 was delivered as a single-user system. Yet as the scope of the application grew, it became increasingly clear that multiple users would need to be able to read data from a common database. Locking out HyperTech users who simply wished to browse a hypertext stored in the same database as a hypertext being modified seemed unnecessarily restrictive. With the addition of the Automatic Path Generation batch facility, which required that two users be able to access the same database simultaneously (the human user as well as the Auto-Path batch process), we clearly needed to redesign the database schema for better support of concurrent users.

The final solution supports multiple readers and a maximum of a single writer in a specified database. We allow the user to commit or revert changes in a session at his or her discretion, just as with the single-user design. At the same time, we maintain data consistency and do not permit readers to access uncommitted data in use by an active writer.

This solution was complicated by Sybase's concurrency control mechanism, a locking scheme which ensures consistency of data being accessed by multiple users. While logically, HyperTech could restrict access to a certain hypertext at a time, the physical mapping of HyperTech objects onto records in Sybase database tables makes that restriction impossible to enforce due to Sybase's implementation of locks.

Specifically, the main objects in HyperTech are hypertexts, whose contents are stored in multiple records across multiple tables in the database. HyperTech assembles and disassembles the hypertext components from these records, and the user is never aware of the composition and decomposition happening on the fly. Unfortunately, Sybase's locking scheme does not provide the record level control we required, i.e., it does not allow us to restrict access to all records related to a given hypertext. Instead Sybase implements a page level lock or a table level lock when modifying data in a table. Operations which intend to modify a single record (such as the act of renaming a node) would, at a minimum, lock all the records on the data page containing that record. Thus the system restricts access to records which might have nothing to do with the hypertext being modified.

Several different work-arounds were attempted before we arrived at the approach implemented in HyperTech 2.0. The first attempt was to install a time-out mechanism, so that users would time-out when trying to access locked data. HyperTech would notify users when the time-out occurred, allowing them to retry or abandon the operation. This approach was infeasible due to general user interface concerns, e.g., a time-out due to server inaccessibility looked the same as a data lock conflict. Additionally, the need to wait for a time-out was very annoying to the user.

The next approach was to force record level locking by placing each record on its own page in the database. A page level lock would then lock only the single record, instead of other unrelated records that were coincidentally stored on the same page. However, we ran into a problem with this solution due to the need for indices. When updating data, if a column is not indexed, then the more sweeping table level lock is applied. Page level locks are only used when indices are in place. When updating indexed records, the indices must be updated, and to update an index, a lock on the page containing the index must be made. Since multiple indices are stored on a page, we still ran into conflicts that locked out entire pages instead of just records.

Since we were unable to "fool" the record-level locking mechanism, we decided to circumvent it entirely through the use of the "dirty-read" facility in Sybase SQL Server v10.1. A dirty read is the act of reading data regardless of any locks in place. But this meant a reader could potentially access uncommitted data in the database, data which a writer might later revert. Thus, we could not ensure data integrity, as a user could access the contents of a node as it was being deleted. Since hypertext objects are stored in multiple records, the deletion operation would take several steps to perform, and the user could end up seeing the text of a node that no longer existed. The requirement of data consistency would be violated by this solution.

We came to the conclusion that there was no way to outsmart the locking mechanism, and thus a database redesign was needed which would avoid lock conflicts as much as possible. The final solution was to implement database versioning, whereby each record is tagged with a version number. Readers examine HyperTech objects formed from the latest published version of the data, i.e., those records with the latest version less than or equal to the value in a special "version" table in the database. Writers create versions which are one greater than the value in this table, using a copy-on-write scheme. With copy-on-write, all modifications are handled by making a copy of the last published record, inserting modifications into this copy, and finally incrementing the database version on the copy. Publishing the version for future readers simply increments the value of the record in the special version table.

This solution has proven to be simple, clean, and relatively easy. Actual modifications to the database, in terms of inserting new records, deleting old records, or modifying existing records for a new version are all performed in short transactions, thereby keeping lock time to a minimum. While it would have been preferable to design for a multi-user environment from the very beginning, we are pleased with the solution developed mid-stream, as we think it is both solid and extensible.

2.2 Use of Xew Widget Set

Finding an existing Motif widget that could support the display of multimedia data and provide WYSIWYG editing features proved to be a challenge. Our goal was to use an existing widget, rather than develop one of our own. The widget needed to support styled text, in a variety of fonts and sizes, along with support for embedding graphics, video and audio data. It also needed to be "Motif-aware". The Motif Text Widget provides none of these features, so we initiated a search for public domain widgets that would provide these features.

After examining a number of possible candidate widgets, none of which satisfied all of the above requirements, we settled on the Text Editor Widget in the Eurobridge Widget Set, commonly referred to as Xew. Xew contains a collection of widgets which were used as the basis for HyperTech's Text View. The only requirement it did not meet was integration with the Motif widget hierarchy. As part of the HyperTech project, we rectified this flaw by adding the necessary code to make the Xew widgets Motif-aware. We then submitted our modifications back to the maintainer of the Xew toolkit, where they have been integrated into the widget set since early 1995.

While the Xew Text Editor Widget suffers from a number of flaws (it does not scale up well to large documents, its cursor detection algorithm is flawed, its support for application-specified text tags is incomplete, and it lacks a search capability), it was our only option short of implementing a Text Editor widget ourselves. Such a task would have been a tremendous undertaking. The development of the Xew Widget set was itself part of a four-year research project, of which the implementation of the Xew Text Widget was a significant portion. Efforts to develop a simple Text Widget would have taken resources away from other aspects of HyperTech development. Accordingly, we feel that the use of the Xew Widget set was our best option at the time. However, we recommend that future follow-ons should again survey the available technology to see if newer and better widgets are available.

2.3 HyperTech Storage Format

HyperTech stores its text data in the native storage format used by the Xew Text Widget. In retrospect, it probably would have been better to store the data in the Hyper Text Markup Language (HTML) format. Direct support for HTML tags would have made it easier to import and export HyperTech data into the World Wide Web, functionality which was later added to the HyperTech design as an Engineering Change Proposal.

There are two reasons why this format was not chosen:

- the relative immaturity of the HTML format at the time of this project's inception (circa early 1994);
- deficiencies in the early versions of the Xew widget set.

When this effort began in 1993, none of us foresaw the tremendous explosion in popularity of the Web. HTML 2.0, the version of HTML most widely supported today, was just being developed. The first version of the popular Netscape™ Web browser had not yet been released when HyperTech development began. While HTML was becoming widely used, it was not yet the de facto standard found on the Net today.

In addition, early versions of the Xew widget set had very limited support for loading or extracting data outside of its own format, which is a subset of an ISO standard popular in Europe. It would have been difficult or impossible to parse and convert HTML formatted text to and from the special control sequences used by the Xew Text Editor widget for representing text styling and data embedding. The current version of Xew provides more control for application-specified tagged data, so some of these earlier restrictions are now removed.

In summary, given the popularity and availability of HTML formatted data, we would strongly recommend that future versions of HyperTech be recoded to store text data directly in HTML for better integration with the World Wide Web.

2.4 Enhancement of Navigation Metaphor

Midway through the project, our internal experience with the link and path metaphor for navigating through hypertexts indicated that it was complex and difficult to use. It was necessary to redesign the navigation facilities to clarify and simplify traversal techniques in HyperTech.

The initial approach for HyperTech 1.0 allowed users to create links by specifying link sources and destinations in two separate steps. When both a source and destination had been set, a dialog would appear asking for the name of the link and the name of the path to which this link would belong. Two separate name spaces were provided, one for links and one for paths, leading to some confusion over which name was which. It was easy to lose track of where you were in the link creation process. In addition, the first development effort was not tasked to create any kind of facility for manipulating paths, so we arbitrarily chose to add new links at the end of a path, which was not always the most desirable course of action. Finally, the interface did not adequately identify when the user was manipulating links versus paths.

When tasked to add a Fisheye View to HyperTech 2.0 (a special view to show the contents of a given path) we employed this opportunity to revise the navigation design and chose to implement links as paths connecting exactly two nodes. Effectively, what had been a link in HyperTech 1.0 became the smallest size path. Instead of specifying link sources and link destinations, we added the concept of an anchor, a spot in a hypertext from which navigation may take place or where bookmarks may be placed. Anchors are created on spans of text or graphic. In addition, every node contains a special anchor which represents the entire node, regardless of the amount of text and non-text data it contains. Paths are now composed of a set of anchors traversed in a particular order.

We then enhanced the requirements for the Fisheye View, now known as the Path Browser, and combined it with a PathList Dialog, which listed all the available paths in the system. The result was a full-fledged path editing facility supporting drag-and-drop placement of anchors in paths. With the Path Browser, users may start navigating from any location on any path. Additionally, users have the option of navigating from any path passing through an anchor displayed in the Structure and Text Views.

While there was considerable work involved in retooling the user interface and data model to support this new metaphor, we believe it has been worthwhile in simplifying the navigational support within HyperTech.

2.5 Data Acquisition

In general we had some difficulty obtaining appropriate data to use for testing HyperTech. multimodal data that were as similar as possible to the data used by military intelligence analysts was required, e.g., graphics and aerial or satellite imagery and textual analysis of that imagery. We envisioned intelligence analysts using the system to examine images, make written analyses, and then link those analyses to portions of the images. We therefore needed a database in which texts were related by time or subject to the images so that we could exploit the path building facility of HyperTech. We obtained some imagery and a small amount of text associated with Griffiss Air Force Base and created a small HyperTech database with these files (the "rome" hypertext included with the release of the system). But this hypertext was too small and uncomplicated to provide a rigorous test of linking capability. Discussions with AFSOC (the intended user of the system) did not solve this problem, as we discovered that AFSOC does not generate its own databases. Rather, they primarily use data generated by others.

Since we were unable to obtain large amounts of data from the military community, we turned to publicly available materials. We eventually found data from a repository of weather information kept at the University of Illinois at Urbana-Champaign (UIUC). The "Weather Machine" site at UIUC archives various case studies of important weather events. In these archives we found the "Storm of the Century", a snowstorm that occurred in March of 1993. These case materials seemed to offer a fairly close parallel to the kinds of data in the military intelligence scenario. Included were satellite imagery of regions of the United States both in the visible and infrared portions of the spectrum; various graphics of weather conditions and forecasts; graphics of temperature, pressure, etc.; and a variety of texts. The texts included weather forecasts as well as newsgroup postings by weather professionals discussing the event. These texts could be linked to the various graphics and imagery. The weather graphics could also be linked to the imagery and vice versa. Most of the files carried dates and timestamps that also made it possible to create temporal paths through the data.

With the HyperTech 2.0 software we have included an example of these data and anchoring and path techniques in the "Simple Storm" database, a subset of the full "Storm of the Century" data. We anticipate continuing to use the full "Storm of the Century" database for testing of the Solaris HyperTech implementation.

2.6 Automatic Path Generation

Automatic generation of paths is one of the features that gives HyperTech an advantage over most other working hypertext systems. We have implemented the path generation algorithm described in earlier design documents and proposals — the goal being to link nodes according to textual similarities at the node and paragraph level. Path generation can function automatically in the background, or in a manual mode requiring user intervention. The algorithm used is similar to that of the WAIS search engines now common on the Internet. HyperTech's version of the algorithm is based upon the work of Prof. Gerald Salton of Cornell University.

Thanks to good planning and design, the implementation of this algorithm was relatively uneventful. The biggest potential problem was solved by adding the multi-user support

discussed in Section 2.1, enabling the path generator to run as a separate user of the Sybase database while the human user is simultaneously logged in. We have tested the implementation on a medium size text (a scholarly monograph) and on the Storm of the Century data. It is important to remember that the algorithm only works on text, not on graphics. Further testing will be necessary to determine what kind of data is most amenable to this path generation method. We anticipate the system will work best on large amounts of textual data with a consistent and possibly specialized vocabulary.

2.7 User Interface Techniques

We faced some difficulties in designing a user interface that behaved as a standard Motif application while providing the specialized support needed for hypertext editing. Although the Xew Text Editor widget provides most of the needed features, its deficiencies still hamper the interface to some extent. For example, Xew provides very limited mechanisms to differentiate anchors in text. HyperTech also lacks a good facility for indicating when anchors overlap in a span of text; again, due to the lack of facilities associated with the underlying Text Editor Widget. Finally, better support is needed for interrupting and/or running lengthy operations such as data export in the background.

Although the HyperTech 2.0 interface is sound, any future HyperTech prototypes should continue to refine the user interface towards making the hypertext concepts easier to understand and use.

2.8 Design of User Interface and Data Model

HyperTech was designed as a set of three software layers to facilitate future ports to different database engines and user interfaces. Those layers are:

- The Data Model Access Layer (database independent);
- The Data Model Implementation Layer (database dependent);
- The User Interface Layer.

The Data Model Access layer contains the Data Model Application Programmer's Interface (API). This layer is independent of the user interface used to display HyperTech data. Thus the user interface to HyperTech can be changed without modification to the Data Model API. All public data types are implemented as opaque types, whose implementation is known only to the Data Model Implementation layer.

The Data Model Implementation layer encapsulates the specifics of the relational database engine used to store and manipulate HyperTech data. It contains the implementation of the Data Model Access Layer API, and the details of the opaque data types visible in the API. A port of the Data Model to use a different database engine requires only a port of this implementation layer.

The User Interface layer makes calls to the Data Model API, but contains no direct dependency on any particular database implementation. The User Interface used to display HyperTech data may be changed without impact to the Data Model. Similarly, the Data Model can be modified without affecting the User Interface.

This approach has worked well and was proven effective in an early porting of HyperTech from a linked list database design to the full Sybase implementation. Any future user interface and database engine ports should be similarly successful due to the clear separation of modules and carefully designed interfaces between them.

2.9 Object-Oriented Software Development

This version of HyperTech was implemented in the C language as a conscious decision. Although the use of an object-oriented language such as C++ makes code implementation and reuse easier for the experienced developer, the initial learning curve is considerable. It takes some time to become fluent in the idioms and techniques of object-oriented development. Given HyperTech's schedule and resources, it made better sense to only employ such object-oriented techniques as could be easily implemented in conventional ANSI C.

Now, three years since the program began, object-oriented programming experience has become more common, and porting to an object-oriented language such as C++ or Java become feasible. Java is particularly attractive, not only for its emphasis on Web integration, but also for its reduced complexity and enhanced features compared to C++. Although HyperTech demonstrates that it is possible to develop an object-oriented system without direct support in the programming language, it is certainly more advantageous to have the support structure built-in.

2.10 Performance Issues

While the contractual requirements for HyperTech 2.0 functionality were fully met, the performance of the HyperTech application still needs improvement. Our experience with the current implementation has shown that hypertexts with many subtrees, but small numbers of siblings at a given level, perform better than shallow hierarchies with many siblings at a single level. The simple database caching scheme developed early in the project has improved performance tremendously, but tweaking the parameters of the cache should yield even better results. Some simple modifications in the Structure View should speed up the rendering of a wide hierarchy with many siblings. It is also a documented deficiency that the Xew Text Widget does not handle large amounts of text well, but if we modified the underlying implementation of the Text View, we would expect improvement in the text editing facility. Other performance bottlenecks can be detected by use of the *Quantify* performance analysis tool, and resolved through recoding of the most CPU-intensive modules.

Some of these issues will be addressed while porting HyperTech to the Solaris environment.

2.11 User Feedback

With limited access to potential HyperTech users, we have made educated guesses as to their needs and attempted to address these needs with a powerful, yet easy-to-use interface which facilitates the creation and manipulation of hypertexts. But lacking a significant number of users, and in conjunction with our difficulties in obtaining realistic data, it is difficult to know if we are truly meeting the needs of the user. We hope to cultivate a community of users for HyperTech, whose feedback on the existing application will be invaluable in guiding future directions for the program.

3. Areas for Potential Future Development

3.1 Enhanced Multi-user Support Features

The versioning approach used to implement multi-user support in HyperTech opens the door to a number of possible enhancements. With the addition of a simple protocol, we could add automatic notification of newly published versions of HyperTech databases to existing readers. This automatic notification could allow users the option of examining the latest version of HyperTech data when they become available. Automatic deletion of old versions (after the latest reader of the old version exits) could be incorporated into HyperTech. We could also examine the feasibility of reducing the scope of the write lock, from a databases perspective down to a hypertext or even a section of a hypertext.

3.2 User Interface Toolkit Alternatives

In the two years since choosing the Xew widget set as the basis for the multimodal Text View, more support for multimedia editing facilities has become available. In particular, the Tcl/Tk widget set has a number of widgets which support WYSIWYG editing of HTML. Future work on HyperTech should include another thorough search for public domain software that could become the basis for a robust and faster Text View.

3.3 Data Storage in HTML

As discussed in Section 2.3, we would recommend future versions of HyperTech store text data in HTML format. Such a modification would simplify the process of importing and exporting data from the World Wide Web.

3.4 User Interface Enhancements

For browsing purposes, we believe it would be useful to have a view which serves as a combination of the Structure View and the Text View. This view should show both the hierarchy contained within a node, along with the text of selected nodes within that hierarchy. Such information is currently available by launching a Text View from a selected node in the Structure View, but once the Text View is launched, it is decoupled from the Structure View from which it was spawned. The combination view would dynamically update its text display as different nodes are selected in the hierarchical display. This view would address concerns about the proliferation of windows launched within HyperTech.

As mentioned in Section 2.7, we would also recommend examining some aspects of the user interface based on feedback from the user community. A number of concerns in the Text View might be addressed by using a different widget as a basis for the Text View. Again, a reexamination of options for the Text View should be included in any future versions of HyperTech.

3.5 Porting to the Java Language

A port of HyperTech to the Java language would be worthwhile in at least three areas. First, a port to a true object-oriented language would facilitate the application of object-oriented techniques to HyperTech development. We could provide a true inheritance hierarchy for the views, and more easily and rigorously enforce the message passing schemes grafted into our C language implementation. Second, a Java-based HyperTech could work with popular Web browsers, such as Netscape™, and provide access to HyperTech and its data in a widely available package. Integrating HyperTech with a Web browser makes HyperTech more accessible to the Web, and the Web more accessible to HyperTech. Finally, such an implementation would be platform-independent, allowing HyperTech browsing and editing on any MS Windows, Macintosh, or UNIX system with a network connection to a database server.

3.6 Performance Enhancements

HyperTech experiences some performance problems as the amount of data in the hypertext(s) increases. However, efforts thus far have been primarily spent on enhancing the functionality of HyperTech, and little attention has been paid to deliberately speeding up the application. As a result, we believe there are some reasonably large performance payoffs possible with only a small amount of reimplementation.

In particular, we would recommend experimenting with some parameters of the database cache to reduce the number of hits to the database directly. We can be more selective regarding some of the calls made to the data model, and try to reimplement the algorithms to work with in-memory data, as opposed to constant retrieval from the database. We can also recode some of the Structure View drawing routines to better handle large hierarchies.

While we intend to improve the speed of HyperTech 2.0 wherever possible during the course of the Solaris port, it would be a worthwhile effort for subsequent versions of HyperTech to exclusively focus effort on performance enhancements, through a thorough analysis of algorithms used and the pattern of database accesses.

APPENDIX B

Summary of Usability Testing Recommendations

RECOMMENDATION: HyperTech users need more clarification of hypertext concepts.

RESOLUTION: Training materials and updated sections of the User Manual were included to address some general confusion about hypertext concepts such as nodes, anchors, paths and hierarchy.

RECOMMENDATION: Reorganize some of the placement and ordering of pulldown menus.

RESOLUTION: The pulldown menus were regrouped and ordered consistently (Control - Edit - View) across all views, which made it easier to predict where those menus were located and what their content should be.

RECOMMENDATION: Some window menu layouts are in need of rearrangement.

RESOLUTION: Several of the views had their internal layout changed slightly to address some concerns raised by the testing. For example, the less-frequently used Overlay Tools palette associated with the Text View is now displayed only on command, and the list of available paths was moved from the right side of the window to the left, where users expected it to be.

RECOMMENDATION: Adequate feedback to the user is lacking in several areas.

RESOLUTION: Various problems with a lack of, or inappropriate, feedback were resolved through some simple re-implementation where possible. Some of the deficiencies were related to the underlying Xew Text widget and cannot be immediately resolved. In the latter case, documentation of the deficiencies was our only recourse.

RECOMMENDATION: The concept of read and write mode must be clarified.

RESOLUTION: To clear up user confusion over whether they were in read or write mode, we changed the name of the modality toggle item from "Toggle Read Only" to "Write Mode". A toggle button appears in the menu if the user is currently a writer of the database. Toggling from write to read-only deselects the toggle button, whereas toggling from read-only to write will select the toggle button (if write permission can be granted). Training the user to look at the title bar of the window to see the current mode (a feature already present) also helps to demystify the problems with modality.

APPENDIX C

Acronym List

AFSOC	Air Force Special Operations Command
ANSI	American National Standards Institute
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
AU	Sun audio format
DTD	Document Type Definitions
FAQ	Frequently Asked Questions
FTR	Final Technical Report
GIF	Graphic Interchange Format
GTRI	Georgia Tech Research Institute
HTML	HyperText Markup Language
IRRE	Image Exploitation Branch
JDBC	Java Database Connectivity
JIT	Just in-Time
JPEG	Joint Photographic Experts Group image format
K	Kilobytes
MPEG	Moving Picture Expert Group video format
OS	Operating System
PNM	Portable Anymap image format
POSIX	Portable Operating System Interface
SGML	Standardized General Markup Language
SQL	Structured Query Language
SunOS	Sun Operating System
TCL/TK	A programming system
TIFF	Tagged Image File Format
UNIX	The UNIX Operating System
UIUC	University of Illinois at Urbana-Champaign
URL	Uniform Resource Locator
WAIS	Wide Area Information Server
W3C	World Wide Web Consortium
WWW	World Wide Web
Xew	EuroBridge multimedia widget set
XIDB	eXtended Integrated Data Base
XML	Extensible Markup Language
XV	An image processing application for UNIX

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.